实验十二　模拟时钟

2020 年秋季学期

杨飞洋 191220138

目录

1. 实验目的：

通过设计一个模拟数字时钟，巩固对之前知识的理解，学会将所学的知识结合在一起做一个产品

2. 实验环境/器材

系统：windows10
开发软件：Quartus 17.1 Lite
开发板：DE10 Standard
仿真环境：ModelSim
芯片：Cyclone V ， 5CSXFC6D6
显示器：标准 VGA 显示器（640*512）
键盘：ps2 键盘

3. 设计思路

本实验设计的模拟时钟实现了以下几个功能：时钟的正常行走，在显示器上绘制表盘，七段数码管显示时间，键盘设置时间、闹钟，整点报时，闹钟响铃功能，功能整合，接下来一点一点谈。

时钟的正常行走

首先时钟最基础的肯定是生成一个周期为 1s 的时钟信号，用一下分频器模块即可实现。

```
module FrequencyDivision(
   input en,
   input clk_input,
   output reg clk_output
);
   reg [31:0] count;
   initial
   begin
      clk_output = 1'b0;
   end
   always @ (posedge clk_input)
   begin
      if (count >= 25000000)
      begin
         count <= 0;
         clk_output <= ~clk_output;
      end
      else if (en)
      begin
         count <= count + 1;
         clk_output = clk_output;
      end
      else
      begin
         clk_output = clk_output;
      end
   end
endmodule
```

这个输出的 clk_output 信号会在总模块中被频繁使用到。

要设置时间和闹钟，那不可能只用一套时分秒的信息，所以在总模块中有以下两个选择键：

```
output wire set_time_en, // 用于进行时间设定的使能端
output wire set_alarm_en, // 同理
```

之所以是 output，是因为我希望做到用键盘来决定这件事，所以就放在键盘模块的输出中了，这个在下面介绍键盘功能时会讲。
有以下两套时分秒的信息，当 set_time_en 和 set_alarm_en 都为 0 时，第一套是有效的，即时钟在正常行走，当 set_time_en == 1 || set_alarm_en == 1 时，则下面一套时间是有效的，具体用哪一套还要看具体的 set_time_en 和 set_alarm_en 的情况。

```
wire [5:0] clock_hour;
wire [5:0] clock_minute;
wire [5:0] clock_second;
//需要用来传递时间的wire变量

wire [5:0] hour_trans;
wire [5:0] minute_trans;
wire [5:0] second_trans;
// 用于传递时间的参数，不仅仅是时间设定的，还可以是闹钟设定的
```

为了之后叙述方便，将上面的称为第一套时分秒，下面的称为第二套时分秒。

接下来是时钟行走模块：

```
module ClockRun(
    input one_second_clk,

    input [5:0] hour_set,
    input [5:0] minute_set,
    input [5:0] second_set,

    input set_en,

    output reg [5:0] hour,
    output reg [5:0] minute,
    output reg [5:0] second
    );
```

输入 1s 的时钟信号，输入设定的时间，将总模块中的 set_time_en 赋给此模块中的 set_en，表示此时正在进行时间设定，需要调整时间。
hour、minute、second 是本模块原生的变量，从 00:00:00 开始行走，只有 set_en=1 时才会改变行为。
以下的代码实现了显示时间的选择和正常的时间行走，即时分秒嵌套进位，分别用 23、59、59 来作为进位信号。

```
if (set_en == 1'b1)
begin
    hour <= hour_set;
    minute <= minute_set;
    second <= second_set;
end

else if (one_second_clk == 1'b1)
begin
    if (second >= 6'd59)
    begin
        second <= 6'b0;
        minute <= minute + 1'b1;
    end
    else
        second <= second + 1'b1;

    if (minute >= 6'd59 && second >= 6'd59)
    begin
        minute <= 6'b0;
        hour <= hour + 1'b1;
    end

    if (hour >= 6'd23 && minute >= 6'd59 && second >= 6'd59)
    begin
        hour <= 6'b0;
    end
end
```

七段数码管显示时间

显示的时间可能是正常行走的时间，也可能是正在设置的时间，也可能是闹钟设定的时间。该模块输入总模块中的两套时分秒和 set_time_en、set_alarm_en 键，两个选择键决定在数码管上输出哪一套时间，输出一个 42 位的变量，用来数码管显示。

```
module TimeShow(
   input [5:0] hour__,
   input [5:0] minute__,
   input [5:0] second__,

   input [5:0] hour_,
   input [5:0] minute_,
   input [5:0] second_,

   input set_en,
   input alarm_en,

   output [41:0] all_hex
   );
```

用一套临时变量来存储信息，不会对上层模块产生影响，比较安全。

```
reg [5:0] second;
reg [5:0] minute;
reg [5:0] hour;
```

接下来还是和之前类似的用选择端来决定输出哪套时间，这里就不多贴代码了。

接下来调用数码管显示模块来显示时间：

```
HEXShow second_show(
    .data(second),
    .hex_one(all_hex[6:0]),
    .hex_ten(all_hex[13:7])
);
```

这里用 secong 数据来解释一下，该模块输入一个数据，返回两个 7 位数据，second 处在 6 个数码管的低两位。用 case 语句来对数据的十位和个位进行两次选择，data/10 的部分就和上面的一样，还是比较容易的。

```
module HEXShow(
   input [5:0] data,
   output reg [6:0] hex_one,
   output reg [6:0] hex_ten
   );

   always @ (*)
   begin
      case (data % 10)
      0:hex_one = 7'b1000000;
      1:hex_one = 7'b1111001;
      2:hex_one = 7'b0100100;
      3:hex_one = 7'b0110000;
      4:hex_one = 7'b0011001;
      5:hex_one = 7'b0010010;
      6:hex_one = 7'b0000010;
      7:hex_one = 7'b1111000;
      8:hex_one = 7'b0000000;
      9:hex_one = 7'b0010000;
      default:hex_one = 7'b1111111;
      endcase
      case (data / 10)
```

在显示器上绘制表盘

这是本实验最有挑战性的部分，在这里我用了一个简易画法。
先来看下本模块的输入和输出：

```
module ClockDrawing(
   input [5:0] hour,
   input [5:0] minute,
   input [5:0] second,


   input clk, // 50MHz
   input rst,
   input reset,

   output hsync, // 行同步和列同步信号
   output vsync,
   output valid, // 消隐信号

   output vga_clk,

   output sync_n,

   output [7:0] vga_r, // 红绿蓝颜色信号
   output [7:0] vga_g,
   output [7:0] vga_b,//都更改为4bits表示
   output reg test
   );
```

输入上层模块中的时分秒信息，输入 50MHz 的时钟信号，置位端，输出 VGA 显示所需要的一系列信号。

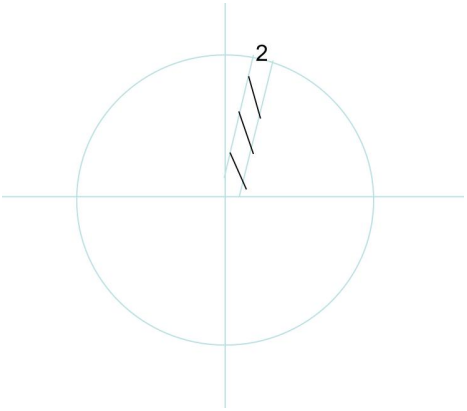
我采用将整个显示屏看成一个坐标系，用**线性规划**的方式来进行着色。
以整个屏幕中心为原点，即(320,256)，因为 VGA 显示屏是 640*512 分辨率的。
由于这是在显示器是绘制的一个模块，那么接下来的 clkgen 和 vga_ctrl 两个模块是一定会被用到的。

```
clkgen #(25000000) clkgen_fuction(
   .clkin(clk),
   .rst(rst),
   .clken(1'b1),
   .clkout(vga_clk)
   );


 vga_ctrl vga(
    .pclk(vga_clk),
    .reset(reset),
    .vga_data(data_trans),
    .h_addr(h_addr),
    .v_addr(v_addr),
    .hsync(hsync),
    .vsync(vsync),
    .valid(valid),
    .vga_r(vga_r),
    .vga_b(vga_b),
    .vga_g(vga_g)
    );
```

上图代码中的 h_addr 和 v_addr 代表现在扫描的位置，不妨令横坐标 x 为 h_addr 和原点的插值，同理令纵坐标 y 为 v_addr 和原点之间的差值

```
x = h_addr[9:0] - 320;
y = - (v_addr[8:0] - 256);
```

现在有了 x 和 y 两个参数，可以开始进行线性规划了。
以分针为例，假如现在分钟指向的是 2，那么下图就是我的作图思想。



首先是 x>0，y>0，然后 x，y 处在一个圆内，我设置这个圆的半径为 200，
则 x*x + y*y <= 40000，然后再用两根直线框住，这个直线的斜率是能够计算的，一格是 6°，那么 2 对应的斜率就为 tan(90-2*6) = tan78° ≈4.704，再选择一下截距，我选择 4，即 4.704x – 4 < y < 4.704x + 4，也即-4000<1000y + 4704x < 4000，我将每个指针位置对应的斜率存进一下数组

```
reg signed [19:0] angle [59:0];
```

之所以选择 signed 型来存储，是因为不大放心浮点数在硬件中的行为，所以就将 tan 值*1000 取个近似值存进斜率数组。
其实只需要算 14 个值就好了，0 和 15 对应的地方都赋为 0，
angle[1] = tan(90-6)* 1000 = 9514
angle[2] = tan(90 – 2*6)* 1000=4704
……
angle[14] = tan(90 – 14*60)*1000 = 105

由于表盘上的对称关系，例如 2 和 31 对应斜率是一样的，则 angle[32] = angle[2]，又例如 2 和 28，58 都是相反的，那么 angle[58] = angle[28] = - angle[2]
那么以下就是存储结果了。

```
angle[0] = 0;angle[30] = 0;
angle[1] = 9514;angle[29] = -9514;
angle[2] = 4704;angle[28] = -4704;
angle[3] = 3077;angle[27] = -3077;
angle[4] = 2246;angle[26] = -2246;
angle[5] = 1732;angle[25] = -1732;
angle[6] = 1376;angle[24] = -1376;
angle[7] = 1110;angle[23] = -1110;
angle[8] = 900;angle[22] = -900;
angle[9] = 726;angle[21] = -726;
angle[10] = 577;angle[20] = -577;
angle[11] = 445;angle[19] = -445;
angle[12] = 324;angle[18] = -324;
angle[13] = 212;angle[17] = -212;
angle[14] = 105;angle[16] = -105; angle[15] = 0;
angle[31] = 9514;angle[59] = -9514;
angle[32] = 4704;angle[58] = -4704;
angle[33] = 3077;angle[57] = -3077;
angle[34] = 2246;angle[56] = -2246;
angle[35] = 1732;angle[55] = -1732;
angle[36] = 1376;angle[54] = -1376;
angle[37] = 1110;angle[53] = -1110;
angle[38] = 900;angle[52] = -900;
angle[39] = 726;angle[51] = -726;
angle[40] = 577;angle[50] = -577;
angle[41] = 445;angle[49] = -445;
angle[42] = 324;angle[48] = -324;
angle[43] = 212;angle[47] = -212;
angle[44] = 105;angle[46] = -105;angle[45] = 0;
```

这个方法虽然比较笨，不过比较容易理解，而且效果也算过得去，最大的缺点就是当斜率大了之后，线性规划导致的截距不平衡就会显示地比较明显。

思路有了，接下来只是用代码把思路陈述下来就行了。

先看**分针**部分的：

```
if(x*x + y*y < 40000)
begin
   if (minute > 0 && minute <= 15)
   begin

      if (1000 * y - x * angle[minute] > -4000 && 1000 * y - x * angle[minute] < 4000 && x > 0 && y > 0)
         data_trans = 16'h00F;
      else
         data_trans = 16'hFFF;
   end
```

以上代码是分钟部分的一小部分代码，当 0<minute<=15 时，点的位置位于第一象限。
if 语句的前两个条件限制了两条直线的位置，线性规划完成。

用这套判断方法可以判断除了 minute = 0 或 30 的其余 58 种情况。
如下：

```
else if (minute > 15 && minute < 30)
begin
   if (1000 * y - x * angle[minute] > -4000 && 1000 * y - x * angle[minute] < 4000 && x > 0 && y < 0)
      data_trans = 16'h00F;
   else
      data_trans = 16'hFFF;
end
else if (minute > 30 && minute <= 45)
begin
   if (1000 * y - x * angle[minute] > -4000 && 1000 * y - x * angle[minute] < 4000 && x < 0 && y < 0)
      data_trans = 16'h00F;
   else
      data_trans = 16'hFFF;
end
else if (minute > 45 && minute < 60)
begin
   if (1000 * y - x * angle[minute] > -4000 && 1000 * y - x * angle[minute] < 4000 && x < 0 && y > 0)
      data_trans = 16'h00F;
   else
      data_trans = 16'hFFF;
end
```

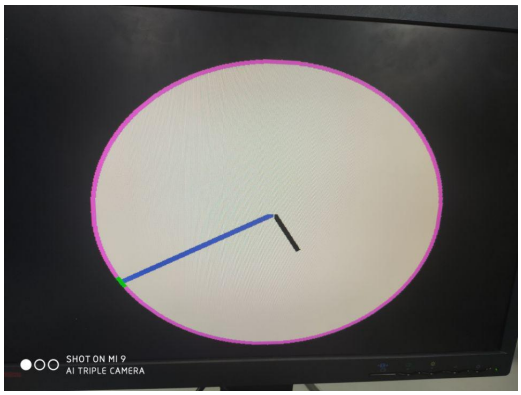
顺带一提，表盘我设计为白色，分针设计为蓝色，就有了 00F 和 FFF 两种情况。
当 minute = 0 或 30 时，1000*y 这个项是会有影响的，所以单独拿出来讨论，其实也很容易，直接手动规定位置就行了，如下：

```
else if (minute == 0)
begin
   if (x < 3 && x > -3 && y > 0)
      data_trans = 16'h00F;
   else
      data_trans = 16'hFFF;
end
else if (minute == 30)
begin
   if (x < 3 && x > -3 && y < 0)
      data_trans = 16'h00F;
   else
      data_trans = 16'hFFF;
end
```

现在还没有解释上面 data_trans 的意义，这是一个 12 位的颜色数据，之后会输送进入 vga_ctrl 模块中。

```
reg [11:0] data_trans;
```

现在来看**秒针**：
我不希望我的表盘上有三根针，这会让整个时钟很臃肿，所以我选择将秒钟以一个小块在圆环是运动的形式展现出来，如下图：



这是我的最终效果图，蓝色的分针比较清晰，紫色圆环上的小绿块就是秒针所指的位置。
秒针的整体思路和分针是一样的，唯二的不同之处在于：
一、在圆环上，那么线性规划时相应改变，我选择半径在 200~205，则
40000 <= x*x + y*y <= 42025
二、如果维持之前的截距的话，当 tan 值比较高时，会很细，图像不明显，所以我把截距调成了 10

秒针 0~15 部分的代码如下，其余和分针的结构一样，就不多赘述了。

```
if  (x*x + y*y >=40000 && x*x + y*y <= 42025)
begin
   if (second > 0 && second <= 15)
   begin
      if (1000 * y - x * angle[second] > -10000 && 1000 * y - x * angle[second] < 10000 && x > 0 && y > 0)
         data_trans = 16'h0F0;
      else
         data_trans = 16'hF0F;
   end
```

时针部分
时针和分针秒针的不同之处是它只有 24 个小时，而不是 60 个，而且很难进行调整，因为这个模块中的时分秒信息是上层模块传下来的，在七段数码管是显示必然是只有 24 个状态，那就设置 24 个状态吧。

和分针的不同：
一、 时针的长度、粗细都有所改变，取半径为 70，即 x*x + y*y < 4900，粗细用截距来调整，我这里调整为 6
二、 hour 和 minute 之间是 5 倍关系，原本的 angle[minute]应改为 angle[hour*5]

时针部分代码：

```
if(x*x + y*y < 4900)
begin
   if (hour > 0 && hour <= 3)
   begin
      if (1000 * y - x * angle[hour * 5] > -6000 && 1000 * y - x * angle[hour * 5] < 6000 && x > 0 && y > 0)
         data_trans = 16'h000;
      else if(data_trans != 16'h00F)//
         data_trans = 16'hFFF;
```

其余位置和分针没有什么差别，只要在时针特定的位置做必要的改动即可。
这里我将时针设为黑色。

到这里，显示器绘制表盘的功能就完成地差不多了。

键盘设置时间

主要思路就是按下 A 时，是设定闹钟的情况，按下 S 时，是设定时间的情况，按下 D 时，退出时间的设定。
设定时间是依次按下 6 个数字，分别对应时分秒的六位数字。

该模块的输入输出：

```
module KeyTimeGet(

    input ps2_clk,
    input ps2_data,

    input CLK_50,

    output reg set_en,
    output reg alarm_en,

    output reg all_ready,
    output reg [5:0] hour,
    output reg [5:0] minute,
    output reg [5:0] second,

    output wire trans_en
);
```

键盘数据是要输进本模块的，还有 50MHz 的时钟信号，输出 set_en 和 alarm_en 两个选择端，分别对应上层模块中的 set_time_en 和 set_alarm_en 位，输出一套时分秒，毫无疑问，这一定对应的是上层模块中的第二套时分秒，即

```
wire [5:0] hour_trans;
wire [5:0] minute_trans;
wire [5:0] second_trans;
// 用于传递时间的参数，不仅仅是时间设定的，还可以是闹钟设定的
```

trans_en 返回给上层模块表示调整有效。

几个变量：

```
wire [7:0] data;
wire ready;
wire [7:0] ASCII;

reg [2:0] location;
```

data 表示键盘码，ASCII 码，location 表示调整数字的位置。

本模块调用 ps2_keyboard 模块和 KeyTranslate 模块

```
ps2_keyboard keyboard(
    .clk(CLK_50),
    .clrn(1'b1),
    .ps2_clk(ps2_clk),
    .ps2_data(ps2_data),
    .data(data),
    .ready(ready),
    .nextdata_n(1'b0)
    //.overflow(overflow)
    );
```

老师提供的 ps2_keyboard 模块可以帮助输出键盘码 data

```
KeyTranslate GetASCII(
    .ready(ready),
    .now_data(data),
    .ASCII(ASCII),
    .CLK_50(CLK_50),
    .out_en(trans_en)
);
```

此模块将键盘码转换成 ASCII 码，ASCII 码存放在一个 txt 文件中，依靠 readmemh 函数读取数据，并且输出一个有效位，表示这是一次有效输入，有效位是依靠断码来判定的，其中需要用多一个变量来存储暂时的 data 数据，具体这个模块里的细节就不多展示了，实验 8 中已做详细叙述。

接下来就是对得到的 ASCII 码进行处理了，其实只要处理 A、S、D、0~9 即可
如果是 A，alarm_en = 1，如果是 S，set_en=1，如果是 D，全置为 0
A 和 D 的情况，都将时分秒信息和 location 位置清零，以便从头开始调整时间。
代码如下：

```
if (alarm_en == 1'b0 && set_en == 1'b0)
begin
    if (ASCII == 8'h61) // 此时是A
        alarm_en = 1'b1;
    else if (ASCII == 8'h73) // 此时是S
        set_en = 1'b1;

    hour = 8'b0;
    minute = 8'b0;
    second = 8'b0;
    location = 3'b0;
end

else if (ASCII == 8'h64) // 此时是D
begin
    alarm_en = 1'b0;
    set_en = 1'b0;
end
```

那如果是 0~9 的情况呢。
此时 set_en 和 alarm_en 必然有一个是为 1 的，所以不必担心乱改。
每改一次 location+1，十位需要*10，比较清晰。

```
else if (ASCII >= 8'h30 && ASCII <= 8'h39)
begin


    case (location)
    0:begin hour = hour + (ASCII - 8'h30) * 8'd10;location = location + 3'b1;end
    1:begin hour = hour + (ASCII - 8'h30);location = location + 3'b1;end
    2:begin minute = minute + (ASCII - 8'h30) * 8'd10;location = location + 3'b1;end
    3:begin minute = minute + (ASCII - 8'h30);location = location + 3'b1;end
    4:begin second = second + (ASCII - 8'h30) * 8'd10;location = location + 3'b1;end
    5:begin second = second + (ASCII - 8'h30);location = location + 3'b1;end
    default:;
    endcase
```

把这个输入的时间返回给上层模块，比如此时 set_en=1，则上层模块中的 set_time_en=1，则 clockrun 模块中改变时间，将这套新的时间赋给第一套时分秒，就达到了调整时间的目的，第一套时间输入给显示器模块，表盘上同时更新时间，做到同步。

闹钟功能和整点报时

闹钟的设置在上面已经讲过了，就是当键盘按下 A，依次输入 6 个数字就行了。
此模块的输入输出：

```
module AlarmRun(
   input [5:0] hour_set,
   input [5:0] minute_set,
   input [5:0] second_set,//用于设定闹钟的变量

   input [5:0] hour,
   input [5:0] minute,
   input [5:0] second,

   input CLK_50,

   input alarm_set,

   input one_second_clk, // 每秒的时钟周期

   output [3:0] LEDR,
   output reg LEDAlarm, // 测试用的LED闹铃输出

   inout                       AUD_BCLK,
   output                      AUD_DACDAT,
   inout                       AUD_DACLRCK,
   output                      AUD_XCK, // 音频实验所需要的大量引脚

   output                      FPGA_I2C_SCLK,
   inout                       FPGA_I2C_SDAT

   );
```

自然输入的是上层模块中的第一套和第二套时分秒和 set_alarm_en 选择端。
输入 1s 的时钟信号，因为我想实现一个 8 秒的闹钟，还有音频实验需要的大量引脚。

```
reg [5:0] alarm_hour;
reg [5:0] alarm_minute;
reg [5:0] alarm_second;

reg [2:0] count;
```

用一套新时分秒来存储闹钟时间，用 count 来记录响的秒数，闹钟维持 8 秒。

这里不仅仅当到达闹钟时间时 count 置为 1，当 minute = 59 且 second = 59 时也置为 1，顺便完成整点报时功能。

```
always @ (posedge one_second_clk)
begin
   if (alarm_set == 1'b1) // 此时是需要设定闹钟的时候
   begin
      alarm_hour = hour_set;
      alarm_minute = minute_set;
      alarm_second = second_set;
   end

   else if ((hour == alarm_hour && minute == alarm_minute && second == alarm_second)||(minute == 59 && second == 59))
   // 此时是闹钟设定的时间和整点报时
   begin
      count = 3'b1; // count加一进行计数
      LEDAlarm = 1'b1;
   end

   else if (count != 3'b0)
   begin
      count = count + 1;
      //LEDAlarm <= 1'b1;
   end

   else if (count == 1'b0)
   begin
      LEDAlarm = 1'b0;
   end
end
```

上面的 LEDAlarm 是用来测试的，可以不管，count 之所以如此重要，是因为它是要被输送进音频模块的，而且 count 的值决定了音频模块输出怎么样的音频信号。

对音频模块的调用：

```
sound_sample Sound(
    .CLOCK_50(CLK_50),
    .LEDR(LEDR),
    .AUD_BCLK(AUD_BCLK),
    .AUD_DACDAT(AUD_DACDAT),
    .AUD_DACLRCK(AUD_DACLRCK),
    .AUD_XCK(AUD_XCK),
    .FPGA_I2C_SCLK(FPGA_I2C_SCLK),
    .FPGA_I2C_SDAT(FPGA_I2C_SDAT),
    .en_count(count)
);
```

看一下这个模块本身：
主要是对一些模块的调用，输入输出的一些细节就不展示了。

```
wire clk_i2c;
wire [15:0] audiodata;
reg [15:0] freq;

initial
begin
    freq = 16'b0;
end

audio_clk u1(CLOCK_50, reset,AUD_XCK, LEDR[3]);


//I2C part
clkgen #(10000) my_i2c_clk(CLOCK_50,1'b0,1'b1,clk_i2c);  //10k I2C clock


I2C_Audio_Config myconfig(clk_i2c, 1'b1,FPGA_I2C_SCLK,FPGA_I2C_SDAT,LEDR[2:0]);

I2S_Audio myaudio(AUD_XCK, 1'b1, AUD_BCLK, AUD_DACDAT, AUD_DACLRCK, audiodata);

Sin_Generator sin_wave(AUD_DACLRCK,1'b1,  freq, audiodata);

always @ (posedge CLOCK_50)
begin
    case (en_count)
    0:freq = 16'd0;
    1:freq = 16'd714;
    2:freq = 16'd802;
    3:freq = 16'd900;
    4:freq = 16'd714;
    5:freq = 16'd900;
    6:freq = 16'd954;
    7:freq = 16'd1070;
    default:freq = 16'd0;
    endcase
end
```

是不是很熟悉呢，和实验九如出一辙呀，唯一的不同就在于 freq 的赋值方式发生了变化，用 case 语句对 en_count 也即上层模块中的 count 的值 switch。

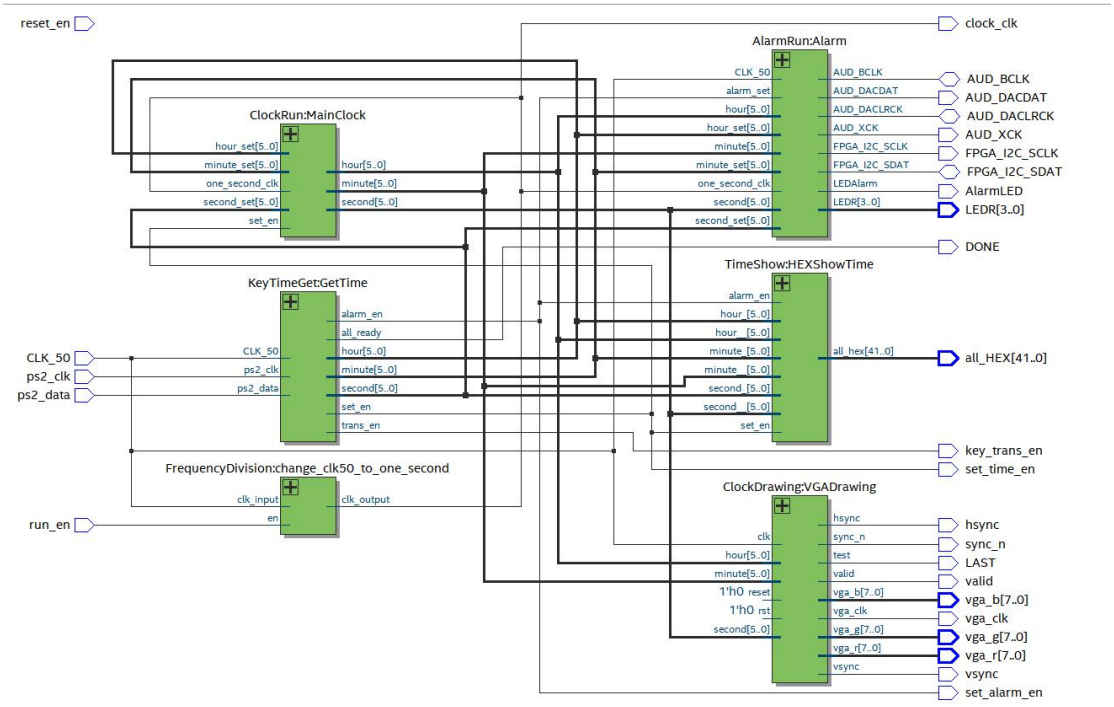
得到了 freq 之后，先用 audio_clk 模块生成 AUD_XCK，clkgen 生成 I2C_CLK，Sin_Generator 模块依靠本模块的 freq 值生成 audiodata，I2S_Audio 模块依靠 audiodata 输出一系列音频实验所需的引脚，I2S_Audio_Config 模块生成 FPGA_I2C_SCLK,FPGA_I2C_SDAT。

这样，闹钟响铃和整点报时功能也完成地差不多了。

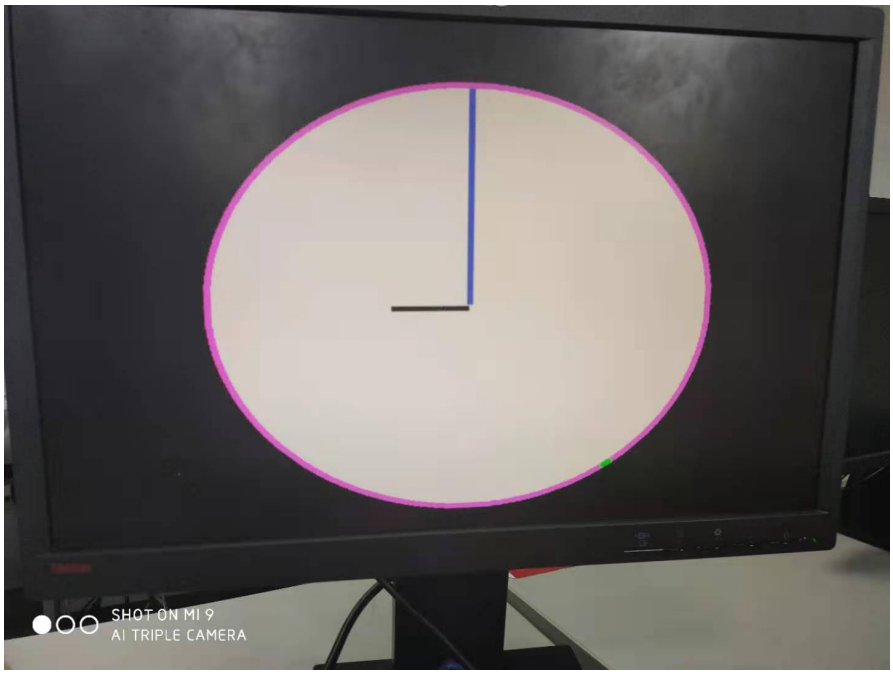
总模块中的模块相互调用：

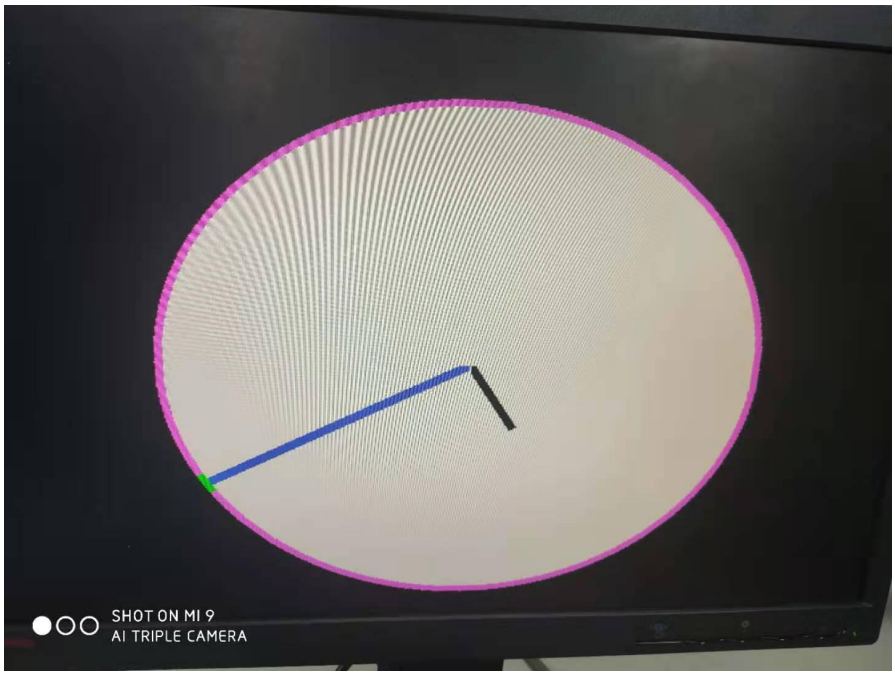
```
//分频器，输出1s的时钟信号
FrequencyDivision change_clk50_to_one_second(run_en,CLK_50,clock_clk);

//主要的时钟行走模块
ClockRun MainClock(clock_clk,clock_hour,clock_minute,clock_second,
    set_time_en,hour_trans,minute_trans,second_trans);

//7段数码管显示时间
TimeShow HEXShowTime(clock_hour,clock_minute,clock_second,
    hour_trans,minute_trans,second_trans,set_time_en,set_alarm_en,all_HEX);

//键盘输入时间
KeyTimeGet GetTime(ps2_clk,ps2_data,CLK_50,set_alarm_en,key_trans_en,
hour_trans,minute_trans,second_trans,DONE);

//闹钟和整点报时
AlarmRun Alarm(hour_trans,minute_trans,second_trans,clock_hour,clock_minute,clock_
set_alarm_en,clock_clk,CLK_50,LEDR,AlarmLED,
AUD_BCLK,AUD_DACDAT,AUD_DACLRCK,AUD_XCK,FPGA_I2C_SCLK,FPGA_I2C_SDAT);

//绘制表盘
ClockDrawing VGADrawing(clock_hour,clock_minute,clock_second,
CLK_50,1'b0,1'b0,hsync,vsync,valid,sync_n,vga_clk,vga_r,vga_g,vga_b,LAST);
```

RTL_viewer:



4.实验结果





还有实验结果的视频，在同级目录下。

5.实验中遇到的问题和解决方案

1.在和之前实验的模块进行耦合时总是会有 bug，问了问同学和助教，更好地理解了那些老师给的模块，才能正常使用。

2.绘制表盘真的挺难的，本来想时针分针还做个形状的，但是要用到图形学的算法，学习了一下，但是都要用到很多的循环，而硬件本身对与循环是很不友好的，所以后来就选择了现在这个简陋版本。

3.在键盘设置时间的模块中，最初的想法是有很多功能按键的，但是按键一多，变量就会变多，同时想要兼顾就很麻烦，最后就采用了现在的版本。

……

……

……

6. 实验得到的启示

1.因为自己的性格比较闭塞，下意识地认为和别人合作不好，所以就决定独自完成这个任务，然而即使选择了这个相对容易的实验，一个人的工作量仍是巨大的，看着别人合作完成 cpu 项目，打从心底羡慕，所以之后要学会合作。

2.曾经一个人在机房面对很难解决的 bug 时真的很绝望，然而还是在同学和助教的帮助之下坚持了下来，完成了这个项目，所以不可轻言放弃。

3 硬件语言 verilog 其实在我看来还是有许多的不足，很多功能不好实现，不过它很好地诠释了硬件是如何工作的，表面写代码，实则连电路。

4.fpga 还是很有意思的。

……

……

……